

Atty. Docket No. YO999-002

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

**Patent Application Transmittal**

Commissioner of Patents and Trademarks  
Washington, D.C. 20231

Sir:

Transmitted herewith for filing Under Rule 1.53(f) is the **Patent Application** of:

Inventor: S.N. Chari, M. Kaiserswerth and J.R. Rao

For: OBLIVIOUS PROXYING USING A SECURE COPROCESSOR

Enclosed are:

- ☒ 9 sheets of drawings.
- ☐ An assignment of the invention to International Business Machines Corporation, Armonk, New York 10504.
- ☐ A certified copy of a \_\_\_\_\_ application.
- ☐ An associate power of attorney.

The filing fee has been calculated as shown below:

	(Col. 1)	(Col. 2)
For:	No. Filed	No. Extra
Basic Fee		
Total Claims 59	-20 =	39
Indep. Claims 10	- 3 =	7
<input type="checkbox"/> Multiple Dependent Claim Presented		

Surcharge-Late Filing Fee or Oath or Declaration

Other Than  
Small Entity

Rate	Fee
	\$760.00
x \$18.00=	\$702.00
x \$78.00=	\$546.00
\$260.00	
Subtotal	\$2,008.00
\$130.00	
TOTAL	\$2,008.00

**Deposit Account Authorization**

- ☒ Please charge Deposit Account No. 09-0468 in the amount of \$ 2,008.00. A duplicate copy of this sheet is enclosed.
- ☒ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 9-0468. A duplicate copy of this sheet is enclosed.
- ☒ Any additional filing fees required under 37 C.F.R. §1.16.
- ☒ Any patent application processing fees under 37 C.F.R. §1.17.

Date: January 8, 1999

Respectfully submitted,

By Louis P. Herzberg  
Louis P. Herzberg  
Patent Agent  
Reg. No. 41,500  
Tel. (914) 945-2885

IBM CORPORATION  
Intellectual Property Law Dept.  
P.O. Box 218  
Yorktown Heights, New York 10598

**DOCKET NUMBER: YO999-002**1       **OBLIVIOUS PROXYING USING A SECURE COPROCESSOR**2       **FIELD OF THE INVENTION**

3       The present invention is directed to the field of  
4       network security protocols. It is more specifically  
5       directed to the field of secure proxying for computing  
6       devices.

7       **BACKGROUND OF THE INVENTION**

8       Network security protocols are used to ensure privacy  
9       and integrity of communication on an open public  
10      network. These protocols are typically intended to  
11      achieve end-to-end security guarantees such that the  
12      communication is private to the entities who establish  
13      the parameters of the secure communication channel.

14     Figure 1 shows an example of a typical secure  
15     communication between a client 10 and a server 11. The  
16     client and the server establish such a secure  
17     connection 12 by first establishing the parameters of  
18     the connection through a handshake. Here, the client  
19     and server can use any of a variety of protocols for  
20     communication. Typically the client establishes a

**DOCKET NUMBER: YO999-002**

1 connection with the server through the transport  
2 protocol. Then they jointly establish the parameters  
3 of the secure connection by negotiating the  
4 cryptographic material, if any, to be used to encrypt  
5 the packets of the communication. Typically during  
6 this negotiation phase the peers in the communication  
7 authenticate themselves using one of several  
8 cryptographic mechanisms. After this negotiation  
9 phase, when the client or the server receives a packet,  
10 it identifies the peer of the secure connection using  
11 the mechanisms of the transport protocol such as socket  
12 number, port number etc. This enables the client to  
13 identify the secure communication parameters to be used  
14 to decrypt or verify the data on the channel used.  
15 Without possession of the cryptographic and other  
16 parameters of this secured communication channel, no  
17 entity can eavesdrop on the contents of the packets.  
18 Thus only the client and server can decrypt data sent  
19 on this channel. This is a normal trust model for  
20 communication between a client and server.

21 In several applications such as web-browsing, the  
22 client and the server wishing to establish a secure  
23 connection, do so through a proxy. There are several  
24 reasons for this split communication. The proxy is  
25 able to log communications initiated by the client. An  
26 example is the case of a proxy implementing the SOCKS  
27 protocol. Since the network security protocol  
28 generally ensures that the communication is private to  
29 the client and the server, the proxy functions only as

**DOCKET NUMBER: YO999-002**

1 a gateway. It does not see unencrypted information  
2 flowing through it, and only relays packets to and from  
3 the client to the server.

4 Figure 2 shows another client-server usage scenario  
5 where the client 10 shown talks to a remote server 11  
6 through a proxy 20. Typical examples of such a  
7 scenario are the cases where the proxy implements the  
8 SOCKS protocol or is a HTTP proxy. In these scenarios,  
9 the proxy is used for several reasons. These include  
10 logging the communications initiated by the client (in  
11 the case of the SOCKS protocol) or using a caching  
12 mechanism wherein the proxy stores commonly requested  
13 information (in the case of the HTTP proxy).

14 In the situation shown in Figure 2, the dashed lines 23  
15 indicate that when the client ID establishes a secure  
16 connection with the server 11, this connection is  
17 tunneled through the proxy 20 such that the packets  
18 that flow as part of the communication protocol are  
19 opaque to the proxy 20. This is achieved, for  
20 instance, by encryption of the data flowing between the  
21 client and the server. The proxy 20 can not undo the  
22 encryptions and any other cryptographic mechanisms used  
23 in the secure connection since it does not possess the  
24 cryptographic material negotiated between client 10 and  
25 server 11 using cryptographic means. In this instance,  
26 the protocol 21 that client 10 uses to communicate with  
27 the proxy 20 is the same as the protocol 22 that the  
28 proxy uses to communicate with server 11. The packets

**DOCKET NUMBER: YO999-002**

1 in the protocol are relayed by the proxy 20 without any  
2 intervention.

3 In some application scenarios, where computationally  
4 limited clients wish to connect over wireless or other  
5 links, the proxy may provide the following additional  
6 functionalities:

7 (a) If the client does not support the same  
8 communication protocols as the server, the proxy acts  
9 as a converter between the protocols the client  
10 supports and those that the server understands. For  
11 example, the client may have the protocols specified by  
12 the Wireless Applications Protocol (WAP), whereas the  
13 server may support only the protocols of the Internet  
14 Protocol (IP).

15 (b) The proxy adapts the content supplied by the  
16 server to fit the constraints of the communication  
17 links and the computational, functional and graphical  
18 constraints of the client.

19 Figure 3 shows an example of an application scenario  
20 where the client 10 does not support the same  
21 communication protocols as the server 11. Thus, for a  
22 communication link to be established the client  
23 communicates to a proxy 20 using a set of protocols 30  
24 which both the client and proxy support. The proxy 20  
25 then communicates to the server 11 using a different  
26 set of protocols 31 which the server 11 supports. If

**DOCKET NUMBER: YO999-002**

1 the client 10 wishes to establish a secure connection  
2 to the server 11, it can only do so by establishing a  
3 secure connection to the proxy which then establishes a  
4 second connection to the server. It is important to  
5 note that in such scenarios, the trust model of Figure  
6 1 does not hold. This is because the proxy 20 has to  
7 translate protocol 30 to protocol 31 and vice versa,  
8 and is thus able to see all the messages that flow  
9 between the client and the server. The scenario  
10 depicted in Figure 3 occurs, for instance, if the  
11 client only supports the protocols specified by the  
12 Wireless Applications Protocol (WAP), whereas the  
13 server only supports those of the Internet Protocol  
14 (IP). A typical client is a pervasive computing  
15 device. In this example, due to the considerable  
16 differences in the protocols on either side, the proxy  
17 has to decrypt the data exchanged by the client and the  
18 server in order to forward it to the other side of the  
19 connections.

20 These requirements of protocol conversion and content  
21 adaptation are orthogonal to the end to end security  
22 requirements which are: to prevent exposure and  
23 modification of content by a third party. It is  
24 therefore advantageous to have a system to provide the  
25 original end to end security guarantees without  
26 modifying the server at the back end. In this way  
27 secure communication is possible even without forcing  
28 the client and server to support a common set of  
29 protocols.

1    **SUMMARY OF THE INVENTION**

2    It is therefore an aspect of the present invention to  
3    provide a method, apparatus and system wherein the  
4    server embeds at the site of the proxy an application  
5    in a coprocessor which performs the functions expected  
6    of a proxy without requiring modifications to either  
7    the client or the server. This is particularly  
8    advantageous when the coprocessor is a secure  
9    coprocessor which is a hardware device designed to  
10   survive in hostile environments where it may be subject  
11   to a variety of security attacks. These attacks may  
12   range from attacks to obtain secret information stored  
13   on the coprocessor to attacks used to subvert the  
14   normal functioning of an application executing on the  
15   secure coprocessor. One example of a device which is  
16   particularly suited for use in the system implementing  
17   the invention is a device which conforms to known  
18   tamper resistant standards guidelines. Although, the  
19   invention is particularly useful for secure proxying  
20   involving pervasive computing devices, it is also  
21   advantageous for secure proxying in general computing  
22   devices.

23   In an aspect of the present invention, the invention  
24   enables embedding an application inside a secure  
25   coprocessor at the site of the proxy to perform  
26   functions such as content adaptation and protocol

**DOCKET NUMBER: YO999-002**

1 translation. These may be achieved with varying  
2 degrees of the following guarantees:

3 (a) Given that a certain functionality is  
4 intended and expected of the application executing on  
5 the secure coprocessor, an external agent can neither  
6 subvert nor disrupt the execution of such an  
7 application.

8 (b) In typical network security protocols  
9 consisting of a client and a server, the trust model  
10 guarantees that an untrusted third party can neither  
11 eavesdrop nor subvert the communication between the  
12 client and the server.

13 In one embodiment, the same guarantees of such a trust  
14 model are obtained even though communication occurs  
15 indirectly through the application executing in the  
16 secure coprocessor at the site of the proxy.

17 (c) In some alternate embodiments, the same trust  
18 guarantees are obtained by varying levels of modifying  
19 the server to support the same protocols as the client.  
20 An advantage provided by the method of the present  
21 invention is that it is advantageous that no  
22 modification be required at the client or the server  
23 even if they do not support the same set of protocols.

24 Another aspect of the present invention is to provide a  
25 means by which a server can securely delegate its



**DOCKET NUMBER: YO999-002**

1 functions to a coprocessor located in the networking  
2 infrastructure. In a symmetric fashion, the invention  
3 is used as a means for the client to delegate its  
4 functions to the networking infrastructure. Thus in an  
5 embodiment the invention provides a mechanism where  
6 multiple parties (several clients and servers) use a  
7 secure coprocessor in the infrastructure to enforce an  
8 arbitrary trust model without modifications to the  
9 clients or the servers.

10 **BRIEF DESCRIPTION OF THE DRAWINGS**

11 These and other objects, features, and advantages of  
12 the present invention will become apparent upon further  
13 consideration of the following detailed description of  
14 the invention when read in conjunction with the drawing  
15 figures, in which:

16 FIG. 1 shows a typical secure communication between a  
17 client and a server;

18 FIG. 2 shows a typical usage scenario where the client  
19 shown talks to a remote server through a proxy;

20 FIG. 3 shows example application scenarios in which a  
21 pervasive computing client does not support the same  
22 communication protocols as the server;

**DOCKET NUMBER: YO999-002**

1 FIG. 4 shows an example of the oblivious proxying  
2 scheme in accordance with the present invention;

3 FIG. 5 shows an example of how an oblivious proxying  
4 scheme is used to splice secure connections between a  
5 client and a server in accordance with the present  
6 invention;

7 FIG. 6 shows an example of a secure store used by a  
8 splicing proxy application to store various information  
9 externally without compromising the security and  
10 integrity of the information stored externally or  
11 violating the trust model;

12 FIG. 7(a) shows an example of components of a system  
13 to implement the present invention which runs on a  
14 proxy;

15 FIG. 7(b) shows an example of a flowchart of software  
16 executing on a secure coprocessor in accordance with  
17 the present invention; and

18 FIG. 8 shows an example splicing together two  
19 connections established using the Secure Sockets Layer  
20 (SSL) protocol in accordance with the present  
21 invention.

22 **DETAILED DESCRIPTION OF THE INVENTION**

**DOCKET NUMBER: YO999-002**

1 Figure 4 shows an example of a system to implement a  
2 method of the present invention to ensure that the  
3 trust model between the client and the server is not  
4 violated even in the presence of a proxy. In this  
5 figure, the proxy 20 contains a secure coprocessor 40  
6 as shown. The secure coprocessor is generally  
7 implemented in hardware and is designed to survive in  
8 hostile environments. A client 10 wishing to establish  
9 a secure connection to a server 11, establishes a  
10 secure connection 30 which terminates at the proxy. A  
11 software component which resides on the proxy forwards  
12 the packets to the application in the secure  
13 coprocessor 40 without encryption/decryption. The  
14 coprocessor 40 processes the packet by decrypting it,  
15 performs transformations on the contents as dictated by  
16 protocols 30 and 31, and then forwards the contents  
17 along a secure connection 31 that it has established to  
18 the server. In one embodiment the system includes the  
19 secure coprocessor 40 and two pieces of software. One  
20 piece of software runs inside the coprocessor 40 and  
21 one runs outside on the proxy 20. Figures 5 , 6, 7(a)  
22 and 7(b) show the system in greater detail. In  
23 particular, Figures 7(a) and 7(b) show flowcharts of  
24 the functioning of the different software components.  
25 Although, the invention is particularly useful for  
26 secure proxying involving pervasive computing devices,  
27 it is also advantageous for secure proxying in general  
28 computing devices.

**DOCKET NUMBER: YO999-002**

1 It is particularly advantageous when the coprocessor  
2 shown in Figure 4 offers a high level of security  
3 against a variety of attacks which can range from  
4 physical or other attacks designed to extract secret  
5 cryptographic material stored in the coprocessor, to  
6 attacks designed to subvert the normal functioning of  
7 an application executing on the secure coprocessor. It  
8 is particularly advantageous to use prescribed  
9 guidelines known to those skilled in the art to achieve  
10 a desired level of security against physical and other  
11 attacks.

12 A coprocessor, typically, is a device which can be  
13 added as a peripheral device to a computer. It  
14 communicates with the host computer using either the  
15 Peripheral Component Interconnect (PCI) or the Personal  
16 Computer Memory Card International Association (PCMCIA)  
17 interfaces. Typically, they are designed to offer  
18 tamper resistance through a variety of means including  
19 physical tamper resistant shields and software  
20 components which guarantee the integrity of  
21 applications running inside the coprocessor. Since  
22 they are typically designed for use in high security  
23 environments they offer, either through software or  
24 hardware, a wide variety of cryptographic primitives  
25 which are also used in common protocols for secure  
26 communication.

27 In order to implement the scheme described in our  
28 invention, the server and/or the client, according to  
29 the dictates of their/its security policy, designs one

**DOCKET NUMBER: YO999-002**

1 or more of the applications described and physically  
2 embeds the coprocessor along with the application at  
3 the site of the proxy. A coprocessor can be used in a  
4 wide variety of configurations depending on the  
5 security model that one needs to implement for a  
6 client-server application. For instance, the  
7 coprocessor could be deployed at the site of the proxy  
8 where the proxy is not trusted by either the client or  
9 the server but the tamper resistance of the coprocessor  
10 makes it impervious to security attacks by the proxy  
11 (or any other party) thereby enabling the client and  
12 the server to trust the application executing in the  
13 coprocessor.

14 Another issue relevant to the method of the present  
15 invention is a scheme for the client to discover a  
16 particular proxy which can securely splice connections  
17 to a given server with which the client wishes to  
18 securely communicate. Various embodiments are used  
19 which depend on the actual transport mechanisms that  
20 are used in the communication between the client and  
21 the proxy and those between the server and the proxy,  
22 there are several schemes to achieve this. For  
23 instance, if the transport mechanisms are based on the  
24 Internet Protocol (IP) this is accomplished by the use  
25 of special ports on the secure coprocessor which are  
26 reserved for the particular server. Although the issue  
27 of client finding the right proxy to communicate  
28 securely with a given server is a very important one,  
29 the invention is described without considering this

**DOCKET NUMBER: YO999-002**

1 issue. This is because the system implementing the  
2 invention is assumes that the proxy which splices  
3 connections to the server is known to the client. It  
4 should be apparent to those skilled in the art that  
5 given a particular transport protocol, any available  
6 mechanism may be used to discover the particular proxy  
7 which can securely splice connections to a particular  
8 server.

9 Figure 5 expands on Figure 4 to show an embodiment of  
10 the invention used in a scenario which requires a proxy  
11 20 to splice secure connections between a client 10 and  
12 a server 11. The secure coprocessor 40 contains a  
13 splicing proxy application 50. This application 50 is  
14 designed to translate between secure connections  
15 established with protocol-1 30 and a secure connection  
16 established with protocol-2 31. It is important to  
17 note that the functionality expected of the proxy is  
18 done by the splicing application 50 on the secure  
19 coprocessor 40. There is no change to either the  
20 server 11 or to the client 11. The usual trust model  
21 which guarantees that no one other than the server 11  
22 and the client 10 is privy to the contents is enforced  
23 since a third party can not subvert the application 50  
24 running inside the coprocessor 40.

25 Figure 6 shows another embodiment of the splicing proxy  
26 application from Figure 5. In this figure, in addition  
27 to the components described in Figure 5, there is a  
28 secure store 60 used by the splicing proxy application

**DOCKET NUMBER: YO999-002**

1 50. In order for the splicing proxy application 50 to  
2 splice two secure connections, it stores various  
3 information in the secure store 60. The information  
4 generally includes such things as the current state of  
5 the connections, the parameters negotiated in these  
6 connections, the state of the translation process etc.  
7 Since a secure coprocessor is a typically a small  
8 device, it has limited memory and can not store this  
9 information for all the connections. It is  
10 advantageous for this information to be stored  
11 encrypted using any of the common encryption  
12 algorithms, with a key unknown to the proxy 20. This  
13 store 60 is either co-located with the proxy 50 on the  
14 same machine or it may reside on a different machine.  
15 The proxy 20 uses an information retrieval mechanism 61  
16 to retrieve information from the store.

17 Figures 7(a) and 7(b) show flowcharts for an example  
18 embodiment of the system components to implement our  
19 invention. Figure 7(a) shows a flowchart, usually  
20 implemented in software, of a component which executes  
21 on the proxy. This component is intended to implement  
22 the transport related functionalities and also to  
23 interface between the application running in the  
24 coprocessor and the parties to the communication.  
25 Figure 7(b) describes the functionality implemented by  
26 the application residing in the secure coprocessor.

27 In these figures, the protocols between the client and  
28 the proxy and those between the proxy and the server

**DOCKET NUMBER: YO999-002**

1 are not explicitly described and the flowcharts  
2 describe how to implement generic protocol splicing.  
3 There are steps performed in these flowcharts which are  
4 dependent on the actual communication protocols between  
5 the client and the proxy and those between the proxy  
6 and the server. Although the description of these  
7 steps is done in a generic manner, later we will  
8 indicate two instantiations of these steps to splice  
9 concrete protocols such as the Secure Sockets Layer  
10 (SSL) protocol of the Internet Protocol suite and also  
11 the protocols of the Wireless Application Protocol  
12 suite. In the description of the steps to implement  
13 the invention we do not consider any content  
14 transformations other than formatting the data in the  
15 packets of one protocol to that of the other protocol.  
16 It should be apparent to those skilled in the art that  
17 with simple modifications of the software components in  
18 the secure coprocessor described in the figure one can  
19 implement more general content transformations.

20 As shown in Figure 7(a), the component running on the  
21 proxy essentially functions as the interface of the  
22 application running inside the coprocessor to the  
23 external world. Its functionality is to open transport  
24 connections to the client and the server, read and  
25 write values to the secure store on behalf of the  
26 coprocessor application and to forward packets along  
27 the two connections.



**DOCKET NUMBER: YO999-002**

1 Step 710 describes the beginning of the splicing  
2 operation. It starts when the proxy software receives  
3 a connection request from the client at the transport  
4 level. For protocols such as the Transmission Control  
5 Protocol (TCP), the connection request is the usual  
6 socket connect. For unreliable transport protocols  
7 such as the User Datagram Protocol (UDP), this step is  
8 the receipt of the first communication packet from the  
9 client.

10 In Step 711, an entry is created in the storage module  
11 which identifies this client communication. Later, as  
12 directed by the coprocessor application, this entry  
13 will also be bound to the communication to the server.  
14 In fact, the application in the coprocessor often uses  
15 this entry as a mechanism for bookkeeping on the  
16 splicing process. For instance, it uses this to store  
17 the session parameters for the secure communication  
18 between the client and the proxy, and likewise for the  
19 communication between the proxy and the server. In  
20 some other embodiments, such as when the coprocessor  
21 application is splicing communication between  
22 unreliable protocols, or when it wishes to aggregate  
23 packets to perform content adaptation, this entry is  
24 used to store a plurality of packets that are received  
25 or sent so far, partial results of the content  
26 adaptation process etc. In abstraction, we refer to  
27 this entry as the state of the splicing process. Since  
28 exposure of this state may result in violation of the  
29 communication privacy, in general the coprocessor

**DOCKET NUMBER: YO999-002**

1 application will encrypt the state using any of several  
2 cryptographic means.

3 The loop shown in steps 712 through 720 is executed  
4 for every packet that the proxy receives from either  
5 the client or the server. Upon receiving a packet from  
6 either the client or the server as shown in step 712 it  
7 computes an id of the sender of the packet (steps 713  
8 and 714). It then uses transport level mechanisms to  
9 identify the particular communication and retrieves the  
10 state of the current splicing which is stored in the  
11 storage module (step 715). For instance, if the  
12 communication protocols were TCP, then the source  
13 addresses and the port numbers through which this  
14 packet is carried on can be used to retrieve the  
15 particular entry.

16 In Step 715, a 3-tuple consisting of this retrieved  
17 state, the id of the sender and the actual received  
18 packet, is formed and forwarded to the coprocessor  
19 application. In Step 717, the proxy then waits for a  
20 response from the coprocessor application. As we  
21 describe later, in step 754 (Figure 7(b)), upon  
22 receiving the 3-tuple from the proxy, the coprocessor  
23 application generates a response which includes a  
24 directive for the proxy application, based on the  
25 3-tuple and the requirements of the splicing process  
26 for these communication protocols.

**DOCKET NUMBER: YO999-002**

1 In Step 718, the proxy application executes the  
2 directive in the response generated by the coprocessor  
3 application. For generic communication protocols, the  
4 following are examples of the responses and directives  
5 within these responses. Since we have abstracted away  
6 details of the actual communication protocols in this  
7 embodiment, we only discuss possible generic responses.  
8 Examples include:

9 (a) Terminate the splicing process: This response  
10 is typically generated when the coprocessor wishes  
11 to terminate splicing of this communication. This  
12 ends the proxy application for this client as shown  
13 in step 721.

14 Note that the proxy handles a plurality of clients, in  
15 parallel, using the same steps shown in FIG. 7(a).

16 (b) Open connection to the server: This directive  
17 is to open a transport connection to the server.

18 (c) Forward one or more packets to either client or  
19 server.

20 (d) Close the transport connection to client and/or  
21 the server.

22 (e) NULL response where no further processing is  
23 required for this packet.

**DOCKET NUMBER: YO999-002**

1 In alternate embodiments, the coprocessor application  
2 may generate a plurality of responses which include  
3 directives. In such embodiments, the proxy application  
4 will execute each of the directives sent by the  
5 coprocessor application.

6 After generating the response with the directive, the  
7 coprocessor application typically computes a new state  
8 based on the previous state and the processing it  
9 deemed fit for this packet. If the response 719 is not  
10 to terminate, a new entry is passed back to the proxy  
11 and in step 720 the proxy application replaces the  
12 entry in the storage module with this new entry. If  
13 the response 719 is to terminate, the process ends 721.

14 Figure 7(b) describes the functionality implemented by  
15 the coprocessor application. In this embodiment we  
16 only show how to implement generic protocol splicing.  
17 It should be apparent to those skilled in the art, how  
18 with modifications, a similar application can be  
19 designed to perform content transformations.

20 The sequence of steps executed by the coprocessor  
21 application are as follows. In step 750 it starts with  
22 the receipt of the 3-tuple which is sent by the proxy  
23 application as described in step 716. This 3-tuple is  
24 the id of the sender of the packet (client or server),  
25 the state (which as we described is usually encrypted),  
26 and the packet which was received by the proxy. In  
27 step 751, the coprocessor retrieves the encryption key,

**DOCKET NUMBER: YO999-002**

1 if any, which is used to encrypt the state of the  
2 splicing process. In Step 752, it decrypts the  
3 encrypted state component of the 3-tuple using the key  
4 retrieved in step 751 to obtain T, which is the actual  
5 state of the splicing process.

6 As discussed before, T typically contains session  
7 parameters associated with the secure communication  
8 along the two links and any other state information  
9 that is deemed relevant to the splicing of the two  
10 protocols. Using the session parameters, depending on  
11 the protocol, in step 753, the coprocessor application  
12 may decrypt the encrypted packet P. In Step 754,  
13 depending on the protocols spliced, the previous state  
14 information in T and the new packet that was decrypted  
15 in step 753, the coprocessor application generates a  
16 response including a directive for the proxy  
17 application. Possible responses may include one of the  
18 following directives:

19 (a) Forward a newly computed packet Q to either the  
20 client or the server: This response is used in the  
21 simple case when the proxy application is used to  
22 splice similar protocols and the splicing process  
23 only involves decrypting the packet received in step  
24 753 and re-encrypting the packet according to the  
25 session parameters of the other link. In this  
26 application, the state information T could contain  
27 the session parameters of both links.

**DOCKET NUMBER: YO999-002**

1 One particular embodiment is when the coprocessor  
2 application is splicing two similar protocols, for  
3 example when the communication between the client and  
4 the proxy uses the TCP protocol secured by the Secure  
5 Sockets Layer (SSL) protocol and so is the  
6 communication between the proxy and the server. In  
7 this case, the first few packets from the client to the  
8 proxy are used to negotiate the session parameters  
9 (also known as the handshake packets) for secure  
10 communication as dictated by the SSL protocol  
11 [SSL}. In this stage, the newly computed packet is the  
12 response to the handshake packet received. On  
13 receiving the first handshake packet, the coprocessor  
14 application also generates a response which includes  
15 the directive of opening a TCP connection to the server  
16 and another response which includes the directive of  
17 sending the initial handshake packet to the server.  
18 After the handshake phase, when the session parameters  
19 are committed to, the coprocessor application stores  
20 the session parameters in the state. Once the  
21 handshake phase is complete, the server and the client  
22 send encrypted packets to the proxy and upon receiving  
23 each packet from the sender, the coprocessor  
24 application decrypts in step 754 using the appropriate  
25 session parameters, and computes the new packet Q which  
26 is the decrypted packet, now encrypted with the session  
27 parameters of the other link. The directive in this  
28 case, is to forward this to the receiver.

**DOCKET NUMBER: YO999-002**

1 (b) NULL response: This response is used, for  
2 example, when the coprocessor application wishes to  
3 aggregate a number of packets before processing or  
4 if the dictates of the communication protocols do  
5 not prescribe any possible action at this instant.  
6 In this case as we shall see in step 755, the  
7 coprocessor may simply aggregate the received packet  
8 into the state T.

9 For example, one possible embodiment is the case where  
10 the coprocessor application is performing the function  
11 of splicing between the protocols of the Wireless  
12 Application Protocol (WAP) suite and those of the  
13 Internet Protocol (IP) suite. WAP is particularly  
14 useful for secure proxying involving pervasive  
15 computing devices. In the WAP suite, in the session  
16 protocols such as WTP, the packets are carried over an  
17 unreliable transport mechanism (UDP). One particular  
18 embodiment is the splicing of the communication between  
19 the proxy and the client which is a transaction  
20 protocol of WAP such as the Class 0 or Class 1  
21 transaction, secured by the dictates of the Wireless  
22 Transport Layer Security, with the communication  
23 between the proxy and the server which is done over the  
24 Transmission Control Protocol secured by the dictates  
25 of the Secure Sockets Layer protocol. In this case,  
26 the packets sent by client to the proxy may arrive out  
27 of order or be lost in the communication link. Thus,  
28 when the coprocessor application decrypts a packet, it  
29 may be out of sequence. In this case, it adds this

**DOCKET NUMBER: YO999-002**

1 decrypted packet to the state and issues a NULL  
2 response. Later, when missing packets arrive, it can  
3 retrieve the sequence of packets it received from the  
4 state T. Thus, reliability as dictated by the  
5 transaction protocols of WTP can be implemented.

6 In another embodiment the coprocessor application  
7 performs adaptation of content from the server.  
8 Typically content adaptations require the proxy to  
9 aggregate several packets before it can perform  
10 transformations. Upon decrypting the packet, if enough  
11 packets have not been received to perform adaptations,  
12 the coprocessor application can add this packet to the  
13 state and generate a NULL response. When enough  
14 packets have been received, it can retrieve all the  
15 previously received packets from the state T and  
16 perform adaptations. At that stage, the coprocessor  
17 application can generate the response of forwarding the  
18 (encrypted) result of the adaptations to the client.

19 (c) Terminate response: used to halt the splicing  
20 process.

21 (d) Open connection to server: Generated upon  
22 receipt of the first packet from the client or the  
23 initial connection request.

24 (e) Close connections to the client and/or the  
25 server.



**DOCKET NUMBER: YO999-002**

1 After generating the response, in step 755 the  
2 coprocessor application computes a new state to be  
3 stored as dictated by the communication protocols and  
4 the embodiment. In step 756, if necessary, it encrypts  
5 this new state. In steps 757 and 758 it forwards the  
6 response including the directive and the new state  
7 respectively to the proxy application.

8 In the embodiments described so far, the trust model  
9 enforced by the coprocessor was secure communication  
10 between a client and server. As depicted in Figure 8,  
11 the techniques mentioned in this invention can be used,  
12 in the abstract, to delegate to the infrastructure the  
13 task of enforcing more general trust models between a  
14 plurality of clients 80 and servers 81, by embedding an  
15 application at the coprocessor 82 at the site of an  
16 untrusted proxy. The figure shows a secure coprocessor  
17 application which is part of the infrastructure and  
18 enforces an arbitrary trust model between the set of  
19 clients and servers. Each client or server can trigger  
20 events expected within this trust model to the  
21 coprocessor based application and this can enforce the  
22 trust model. The secure coprocessor application can  
23 physically be anywhere on the networking infrastructure  
24 and need not be physically resident on any of the shown  
25 clients and servers.

26 It is noted that this invention may be used for many  
27 other applications. It provides a general mechanism  
28 wherein a secure coprocessor is placed in the

**DOCKET NUMBER: YO999-002**

1 infrastructure can enforce any arbitrary trust model as  
2 required by the set of parties to the computation. The  
3 descriptions in the figures are primarily shown for the  
4 case when the trust model requires secure communication  
5 between a single client and server. Although the  
6 description is made for particular arrangements and  
7 applications, the intent and concept of the invention  
8 is suitable and applicable to other arrangements and  
9 applications. For example, the types, sizes and shapes  
10 of the protocols used in the communication between the  
11 client and the proxy, the number of clients and servers  
12 in the trust model and the actual trust model between  
13 these client (s) and server (s) can easily be  
14 accommodated by choosing the right combination of  
15 application on the coprocessor, the content  
16 transformations affected by the application on the  
17 secure coprocessor and the state that is maintained in  
18 the secure storage mechanism. Thus, although  
19 embodiments described refer to a 3-tuple, other  
20 embodiments can generally employ an n-tuple for the  
21 specific communication, where 'n' may be any number  
22 suitable to the specific application, not just three.  
23 It will be clear to those skilled in the art that other  
24 modifications to the disclosed embodiments can be  
25 effected without departing from the spirit and scope of  
26 the invention.

27

**DOCKET NUMBER: YO999-002**

1     **CLAIMS**

2     What is claimed is:

3     1. A method for achieving client to server end to end  
4     security guarantees, the method comprising:

5         employing a proxy between the client and the  
6     server to provide connection links between the client  
7     and the server;

8         embedding a secure coprocessor for use as an agent  
9     of the client and/or the server;

10         the coprocessor acting as a converter between at  
11     least one protocol the client supports and at least one  
12     other protocol supported by the server; and

13         employing respective security protocols of said at  
14     least one protocol and said at least one other  
15     protocol.

16     2. A method as recited in claim 1, wherein the  
17     coprocessor is located at the site of the proxy.

18     3. A method as recited in claim 1, further comprising  
19     the step of the coprocessor guaranteeing that an  
20     application embedded in the coprocessor performs to a  
21     degree of security proscribed by the client and/or  
22     server.

**DOCKET NUMBER: YO999-002**

1 4. A method as recited in claim 1, further comprising  
2 the coprocessor assuring that the proxy can not tamper  
3 with the functioning of the agent.

4 5. A method as recited in claim 1, wherein the client  
5 is a pervasive computing device.

6 6. A method as recited in claim 1, further comprising  
7 the step of adapting content supplied by the client to  
8 fit constraints of the server and/or the connection  
9 links.

10 7. A method for providing secure communications on a  
11 network, the method comprising;

12 securely embedding an agent at a proxy in the  
13 network, and

14 splicing a plurality of secure communication  
15 protocols of different protocol suites into the agent.

16 8. A method as recited in claim 7, wherein the step of  
17 splicing includes splicing a security protocol of the  
18 Wireless Applications Protocol suite (WAP) to that of  
19 the Internet protocol (IP) suite.

20 9. A method as recited in claim 8, wherein the  
21 Wireless Applications Protocol suite is used by a  
22 pervasive computing device.

**DOCKET NUMBER: YO999-002**

1 10. A method as recited in claim 7, further comprising  
2 the agent performing at least one content adaptation  
3 function.

4 11. A method as recited in claim 10, wherein the step  
5 of performing includes maintaining communication  
6 privacy.

7 12. A method as recited in claim 7, further comprising  
8 maintaining a state of a splicing process resulting  
9 from the step of splicing.

10 13. A method as recited in claim 12, wherein the step  
11 of maintaining includes employing a storage device  
12 external to the proxy, and using cryptographic means to  
13 encrypt the state.

14 14. A method for providing network security to a  
15 network employing a proxy, the method comprising:

16 embedding a trusted application in a secure  
17 coprocessor located at the site of a proxy; and

18 delegating to a networking infrastructure a task  
19 of enforcing a trust model.

20 15. A method as recited in claim 14, further  
21 comprising guaranteeing that the application is trusted

**DOCKET NUMBER: YO999-002**

1 to enforce the trust model between at least one server  
2 and a plurality of clients.

3 16. A method as recited in claim 14, further  
4 comprising assuring the tamper resistance of the  
5 application.

6 17. A method for secure communication between a client  
7 and a server employing an untrusted proxy; the method  
8 comprising:

9 embedding a coprocessor at the proxy;

10 the proxy receiving a specific communication  
11 request from a client;

12 the proxy forming an n-tuple for the specific  
13 communication;

14 the proxy forwarding the n-tuple to the  
15 coprocessor;

16 the coprocessor generating a response, including a  
17 directive, to the n-tuple;

18 the coprocessor sending the response to the proxy;  
19 and

20 the proxy implementing the directive.

**DOCKET NUMBER: YO999-002**

1 18. A method of claim 17, wherein the coprocessor is a  
2 secure coprocessor.

3 19. A method of claim 17, wherein the step of  
4 receiving includes:

5       awaiting a connection request from a client;

6       creating an entry in a storage module for the  
7 client;

8       determining a sender of each received packet; and

9       retrieving a stored entry.

10 20. A method of claim 19, wherein the n-tuple includes  
11 a sender id, an entry from a storage module and the  
12 received packet.

13 21. A method of claim 17, wherein the client and the  
14 server can be either a sender or a receiver, and the  
15 step of generating includes employing a first protocol  
16 from the sender to the proxy and a second protocol from  
17 the proxy to the receiver and translating between the  
18 said first and second protocols.

19 22. A method of claim 21, wherein the translating  
20 includes decrypting the received packet as specified by  
21 the security parameters negotiated as per the first  
22 protocol and encrypting the decrypted packet as

**DOCKET NUMBER: YO999-002**

1 specified by the security parameters of the second  
2 protocol.

3 23. A method of claim 21, wherein the translating  
4 includes modifying the received packet to meet  
5 constraints of the receiver and wherein the directive  
6 includes forwarding to the receiver the packet  
7 resulting from the step of modifying.

8 24. A method as recited in claim 23, further  
9 comprising aggregating a plurality of packets into a  
10 group of packets and performing content adaptation on  
11 the group of packets.

12 25. A method of claim 17, wherein the communication  
13 between the client and the proxy employ protocols  
14 specified by the Wireless Application Protocol suite  
15 [WAP].

16 26. A system to control security of a proxy  
17 interconnecting a client to a server, the system  
18 comprising:

19 a secure coprocessor used as an agent of the  
20 client and/or the server; and

21 an application embedded in the coprocessor which  
22 acts as a converter between at least one protocol the  
23 client supports and at least other protocol supported  
24 by the server, wherein the secure coprocessor employs



**DOCKET NUMBER: YO999-002**

1    respective security protocols of said at least one  
2    protocol and said at least one other protocol.

3    27.   A system as recited in claim 26, wherein the  
4    coprocessor is located at the site of the proxy.

5    28.   A system as recited in claim 26, wherein the  
6    coprocessor performs functions to guarantee that an  
7    application embedded in the coprocessor performs to a  
8    degree of security proscribed by the client and/or  
9    server.

10   29.   A system as recited in claim 26, wherein the  
11   coprocessor functions to assure that the proxy can not  
12   tamper with the functioning of the agent.

13   30.   A system as recited in claim 26, wherein the  
14   application embedded in the coprocessor adapts content  
15   supplied by the server to fit constraints of the client  
16   and the connection links.

17   31.   A system as recited in claim 26, wherein the  
18   application embedded in the coprocessor adapts content  
19   supplied by the client to fit constraints of the server  
20   and the connection links.

21   32.   A system for providing network security to a  
22   network employing a proxy, the system comprising:

**DOCKET NUMBER: YO999-002**

1           a secure coprocessor located at the site of a  
2   proxy; and

3           a trusted application embedded in the coprocessor  
4   wherein the coprocessor delegates the task of enforcing  
5   an arbitrary trust model to the application.

6   33. A system as recited in claim 32, wherein the  
7   coprocessor functions to guarantee that the application  
8   is trusted to enforce the trust model between at least  
9   one server and a plurality of clients.

10   34. A system as recited in claim 33, where the  
11   coprocessor functions to assure the tamper resistance  
12   of the application.

13   35. An article of manufacture comprising a computer  
14   usable medium having computer readable program code  
15   means embodied therein for achieving client to server  
16   end to end security guarantees, the computer readable  
17   program code means in said article of manufacture  
18   comprising computer readable program code means for  
19   causing a computer to effect:

20           employing a proxy between the client and the  
21   server to provide connection links between the client  
22   and the server;

23           embedding a secure coprocessor for use as an agent  
24   of the client and/or the server;

**DOCKET NUMBER: YO999-002**

1       the coprocessor acting as a converter between at  
2       least one protocol the client supports and at least one  
3       other protocol supported by the server; and

4       employing respective security protocols of said at  
5       least one protocol and said at least one other  
6       protocol.

7       36. An article of manufacture as recited in claim 35,  
8       the computer readable program code means in said  
9       article of manufacture further comprising computer  
10      readable program code means for causing a computer to  
11      effect the coprocessor guaranteeing that an application  
12      embedded in the coprocessor performs to a degree of  
13      security proscribed by the client and/or server.

14      37. An article of manufacture as recited in claim 35,  
15      the computer readable program code means in said  
16      article of manufacture further comprising computer  
17      readable program code means for causing a computer to  
18      effect the coprocessor assuring that the proxy can not  
19      tamper with the functioning of the agent.

20      38. An article of manufacture as recited in claim 35,  
21      the computer readable program code means in said  
22      article of manufacture further comprising computer  
23      readable program code means for causing a computer to  
24      effect the coprocessor adapting content supplied by the

**DOCKET NUMBER: YO999-002**

1 server to fit constraints of the client and/or the  
2 connection links.

3 39. An article of manufacture as recited in claim 35,  
4 the computer readable program code means in said  
5 article of manufacture further comprising computer  
6 readable program code means for causing a computer to  
7 effect the step of adapting content supplied by the  
8 client to fit constraints of the server and the  
9 connection links.

10 40. A computer program product comprising a computer  
11 usable medium having computer readable program code  
12 means embodied therein for providing secure  
13 communication on a network, the computer readable  
14 program code means in said computer program product  
15 comprising computer readable program code means for  
16 causing a computer to effect:

17 securely embedding an agent at a proxy in the  
18 network, and

19 splicing a plurality of secure communication  
20 protocols of different protocol suites into the agent.

21 41. A computer program product as recited in claim 40,  
22 wherein the step of splicing includes splicing a  
23 security protocol of a Wireless Applications Protocol  
24 suite (WAP) to that of the Internet protocol (IP)  
25 suite.

**DOCKET NUMBER: YO999-002**

1 42. A computer program product as recited in claim 40,  
2 wherein the splicing includes maintaining end to end  
3 security guarantees without a modification at the  
4 server.

5 43. A computer program product as recited in claim 40,  
6 the computer readable program code means in said  
7 computer program product further comprising computer  
8 readable program code means for causing a computer to  
9 effect the step of the agent performing at least one  
10 content adaptation function.

11 44. A computer program product as recited in claim 40,  
12 the computer readable program code means in said  
13 computer program product further comprising computer  
14 readable program code means for causing a computer to  
15 effect the step of maintaining a state of a splicing  
16 process resulting from the step of splicing.

17 45. A computer program product as recited in claim 44,  
18 wherein the step of maintaining includes employing a  
19 storage device external to the proxy, and using  
20 cryptographic means to encrypt the state.

21 46. A computer program product comprising a computer  
22 usable medium having computer readable program code  
23 means embodied therein for providing network security  
24 to a network employing a proxy, the computer readable  
25 program code means in said computer program product

**DOCKET NUMBER: YO999-002**

1 comprising computer readable program code means for  
2 causing a computer to effect the steps of:

3 embedding a trusted application in a secure  
4 coprocessor located at the site of a proxy; and

5 delegating to a networking infrastructure a task  
6 of enforcing a trust model.

7 47. A computer program product as recited in claim 46,  
8 the computer readable program code means in said  
9 computer program product further comprising computer  
10 readable program code means for causing a computer to  
11 effect the step of guaranteeing that the application is  
12 trusted to enforce the trust model between at least one  
13 server and a plurality of clients.

14 48. A computer program product as recited in claim 46,  
15 the computer readable program code means in said  
16 computer program product further comprising computer  
17 readable program code means for causing a computer to  
18 effect the step of assuring the tamper resistance of  
19 the application.

20 49. A program storage device readable by machine,  
21 tangibly embodying a program of instructions executable  
22 by the machine to perform method steps for secure  
23 communication between a client and a server employing  
24 an untrusted proxy, said method steps comprising:

**DOCKET NUMBER: YO999-002**

1           embedding a coprocessor at the proxy;

2           the proxy receiving a specific communication  
3   request from a client;

4           the proxy forming an n-tuple for the specific  
5   communication;

6           the proxy forwarding the n-tuple to the  
7   coprocessor;

8           the coprocessor generating a response, including a  
9   directive, to the n-tuple; and

10          the coprocessor sending the response to the proxy;  
11   and the proxy implementing the directive.

12   50. A program storage device readable by machine as  
13   recited in claim 49, wherein the coprocessor is a  
14   secure coprocessor.

15   51. A program storage device readable by machine as  
16   recited in claim 49, wherein the step of receiving  
17   includes:

18          awaiting a connection request from a first client;

19          creating an entry in a storage module for the  
20   client;

**DOCKET NUMBER: YO999-002**

1           determining a sender of each received packet; and  
2           retrieving a stored entry.

3   52. A program storage device readable by machine as  
4   recited in claim 49, wherein the n-tuple includes a  
5   sender id, an entry from a storage module and the  
6   received packet.

7   53. A program storage device readable by machine as  
8   recited in claim 49, wherein the client and the server  
9   can be either a sender or a receiver, and the step of  
10   generating includes employing a first protocol from the  
11   sender to the proxy and a second protocol from the  
12   proxy to the receiver and translating between the first  
13   and second protocols.

14   54. A program storage device readable by machine as  
15   recited in claim 49, wherein the translating includes  
16   decrypting the received packet as specified by the  
17   security parameters negotiated as per the first  
18   protocol and encrypting the decrypted packet as  
19   specified by the security parameters of the second  
20   protocol.

21   55. A program storage device readable by machine as  
22   recited in claim 53, wherein the translating includes  
23   modifying the received packet to meet constraints of  
24   the receiver and wherein the directive includes



**DOCKET NUMBER: YO999-002**

1 forwarding to the receiver the packet resulting from  
2 the step of modifying.

3 56. A program storage device readable by machine as  
4 recited in claim 55, said method steps further  
5 comprising the step of aggregating a plurality of  
6 packets into a group of packets and performing content  
7 adaptation on the group of packets.

8 57. A program storage device readable by machine as  
9 recited in claim 49, wherein the communication between  
10 the client and the proxy employ protocols specified by  
11 the Wireless Application Protocol.

12 58. A method as recited in claim 1, further comprising  
13 the step of the coprocessor adapting content supplied  
14 by the server to fit constraints of the client and/or  
15 the connection links.

16 59. A method as recited in claim 7, wherein the  
17 splicing includes maintaining end to end security  
18 guarantees without a modification to a server involved  
19 in the communications.

20

1           **OBLIVIOUS PROXYING USING A SECURE COPROCESSOR**

2                           **ABSTRACT OF THE INVENTION**

3    A method, apparatus and system is provided for an  
4    entity to facilitate secure communication between a  
5    client and server even when they do not support the  
6    same set of protocols without violating the trust model  
7    which requires that only the client and server be privy  
8    to the contents of the communication. In an embodiment  
9    this is accomplished by embedding at the site of the  
10   proxy an application running inside a secure  
11   coprocessor which translates between the protocols that  
12   the client supports and those that the server  
13   understands. The invention is also useful for purposes  
14   such as adaptation of content at the site of the proxy  
15   without violating the trust model between the client  
16   and the proxy. In general, the scheme describes  
17   mechanisms to securely delegate to the infrastructure  
18   the ability to enforce an arbitrary trust model between  
19   a set of clients and servers participating in some  
20   computational task.

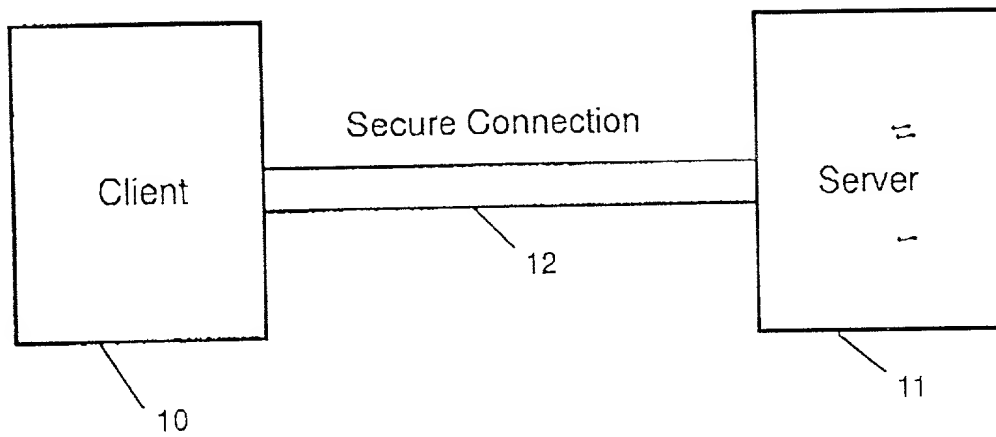


FIGURE 1

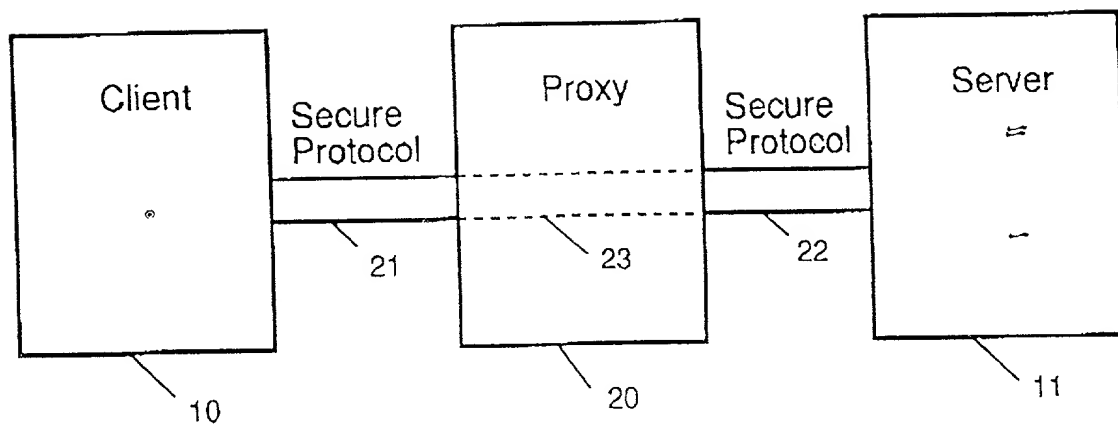


FIGURE 2

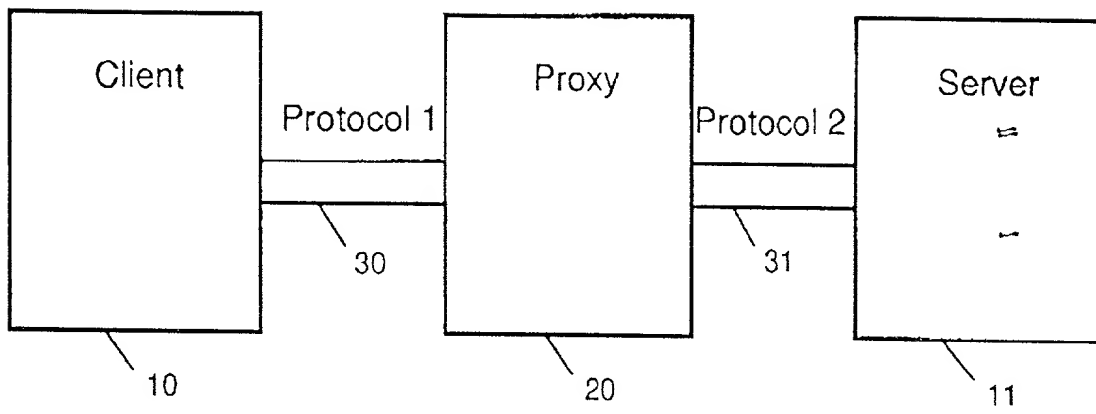


FIGURE 3

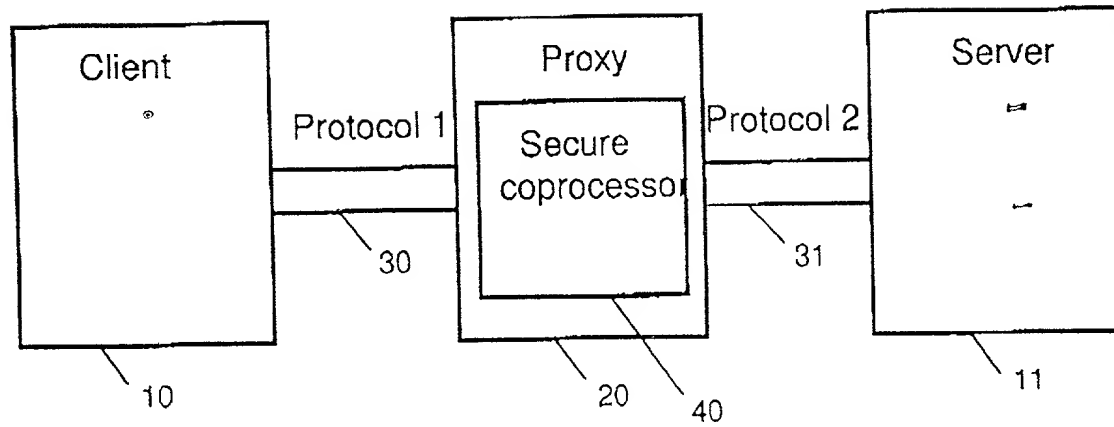


FIGURE 4

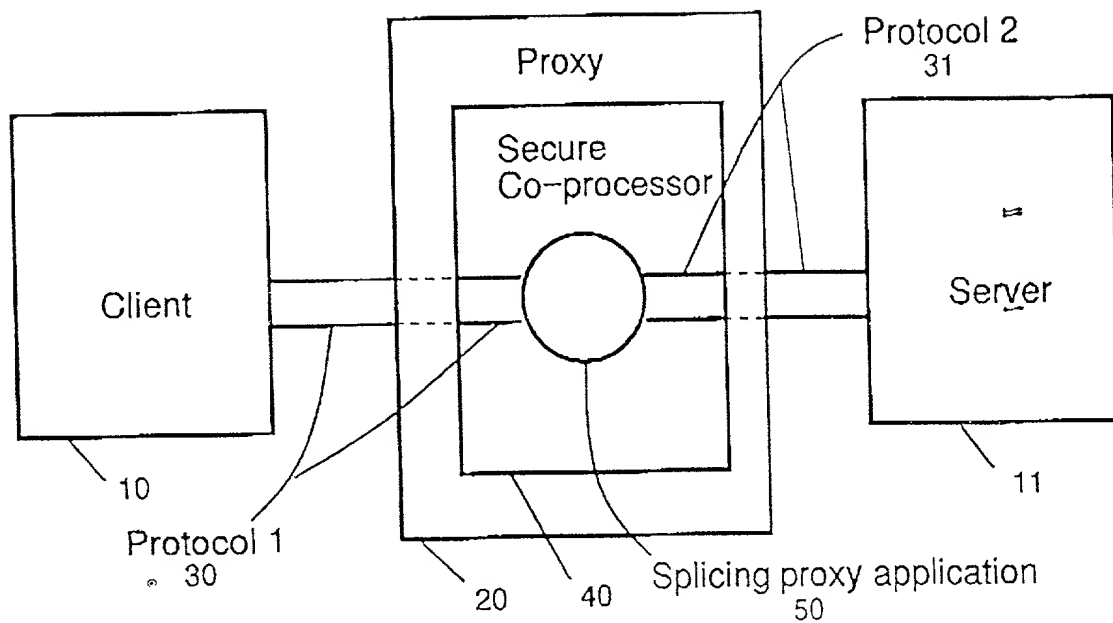


FIGURE 5

6600-00000000

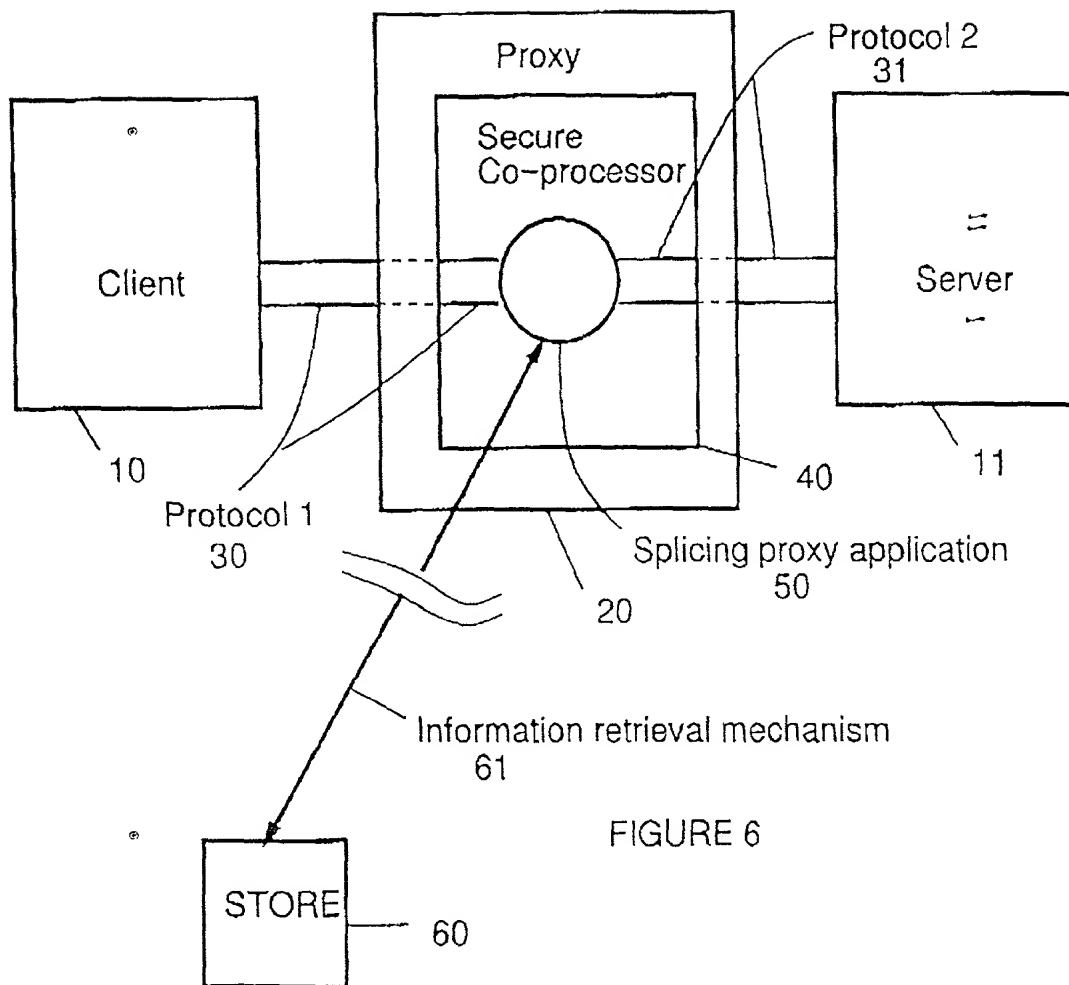


FIGURE 6



• FIGURE 7(a): Description of software running on proxy

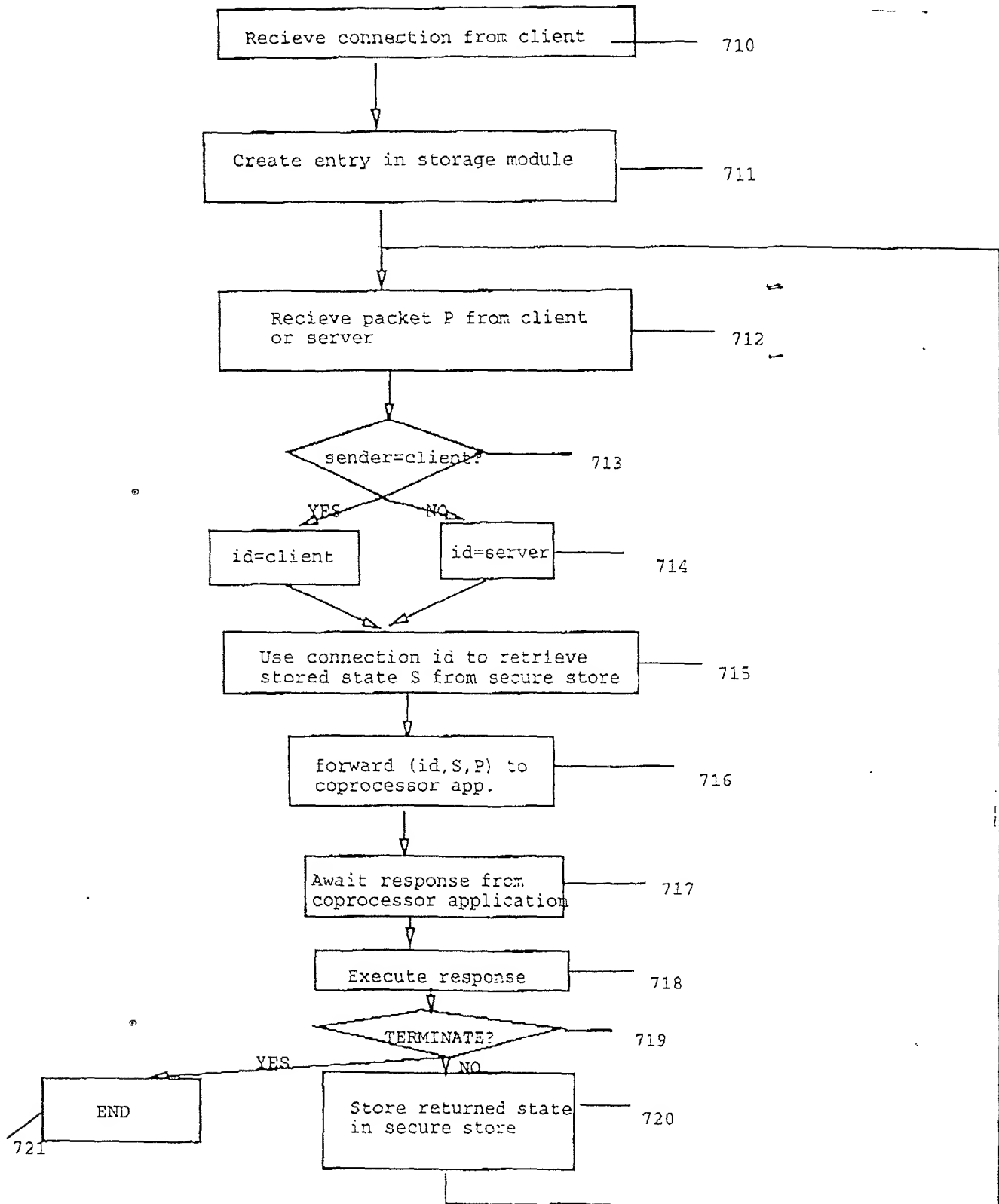
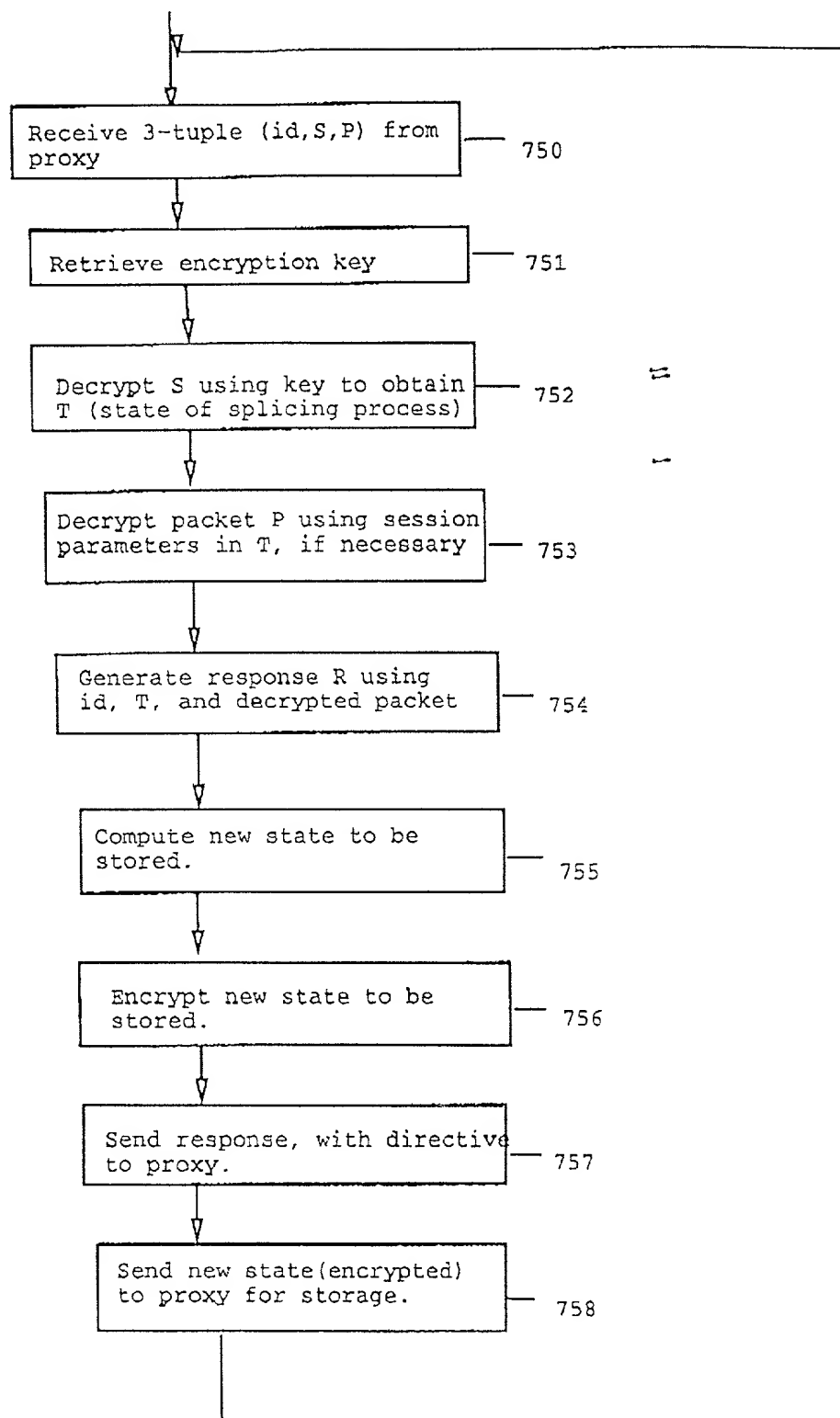


FIGURE 7(b): Description of application process in coprocessor



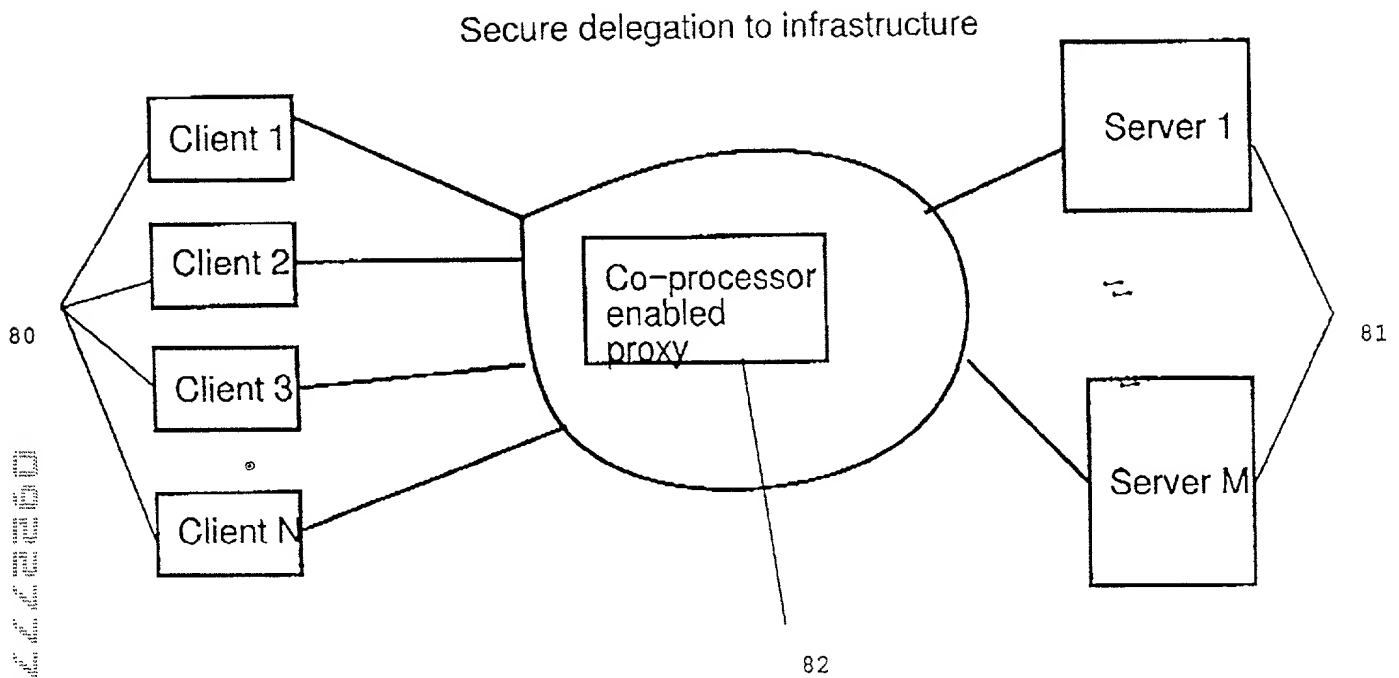


FIGURE 8